Designing automated module testing framework

Jukka Huiskonen

Thesis

Tiko

2011

HAAGA-HELIA
ammattikorkeakoulu

Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

4.9.2011

Tekijä Jukka Huiskonen	**Ryhmätunnus tai aloitusvuosi** 2008
Raportin nimi Moduulitestauksen automaatioympäristön suunnittelu	**Sivu- ja liitesivumäärä** 21 + 20
Opettajat tai ohjaajat Jyri Partanen	

Opinnäytetyön aiheena oli automaattisen moduulitestausympäristön suunnittelu. Työn toimeksiantajana toimi Tellabs Oy.

Ohjelmistotestaus voi jäädä vähemmälle huomiolle, johtuen budjetti tai aikataulullisista syistä, jonka seurauksena asiakkaalle voi päätyä viallinen ohjelmisto. Pahimmassa tapauksessa seuraukset näkyvät negatiivisesti yrityksen liikevaihdossa ja asiakasmäärissä.

Moduulitestauksen automatisoinnilla yritetään siirtää helposti suoritettavat yksinkertaiset testit ihmiseltä tietokoneen hoidettavaksi. Tämän avulla saavutetaan säästöjä kustannuksissa ja nopeutetaan moduulitestausta.

Automaatio moduulitestausympäristön tarkoituksena on tuoda ohjelmistokehittäjille ja testaajille työkalut, joilla voidaan luoda ja ylläpitää automatisoituja moduulitestejä.

Automaatioympäristön suunnittelu jaettiin toteutuksessa kahteen osaan, toiminnalliseen ja arkkitehtuurikuvaukseen. Toiminnallisessa kuvauksessa esiteltiin järjestelmä toiminnalliset ominaisuudet ja määriteltiin niiden vastuut.

Arkkitehtuurikuvauksen pohjana käytettiin edellä mainittua toiminnallista kuvausta. Kuvauksessa listattiin järjestelmän jokainen yksikkö, sen rakenne ja toiminnalliset ominaisuudet.

Toiminnallinen ja arkkitehtuurikuvaus asetettiin salaiseksi Tellabs Oy:n toimesta.

Laadullinen ohjelmistokehitys nojautuu tarkkaan ja harkittuun sekä toiminnalliseen että arkkitehtuurikuvaukseen.

Asiasanat
automaatiotestaus, moduulitestaus

HAAGA-HELIA
University of Applied Sciences

Abstract

4.9.2011

Information Technology Degree programme

Authors Jukka Huiskonen	**Group or year of entry** 2008
The title of thesis DESIGNING AUTOMATED MODULE TESTING FRAMEWORK	**Number of pages and appendices** 21 +20
Supervisor(s) Jyri Partanen	

Thesis focused on the design of automated module testing framework. It was commissioned by Tellabs Oy Automating module testing can make savings in operation costs and increase quality of the developed software.

When developing software with increasing complexity and tight schedule, testing can sometimes be neglected due to time and budget constraints. This can cause severe problems and loss of business if software defects are not found before it is delivered to a customer.

Automated module testing framework is meant to provide developers and testers tool with which to automate those tests that are most often repeated or are most labor-intensive.

Framework design was split into two parts, functional and system architecture specification. Functional specification relied on the requirements and feature requests gathered from developers using meetings and face-to-face conversations.

System architecture specification was constructed from functional requirements by identifying the responsibilities and functions of different modules of the framework.

Result was functional and system architecture specification which was deemed confidential by Tellabs Oy.

Carefully executed and well thought out specification is basis for successful software development.

Key words
automated module testing, framework, module testing

Table on content

1 Introduction

1.1 Assignment

I did my internship at Tellabs Oy from May 2010 to November 2010. My job was to develop module test automation. Automation was seen as a way to decrease the time developers had to use doing manual module testing.

I focused on developing a framework that was designed and implemented in another department. Capture-playback tool was used as a basis for that framework. This was also the main weakness of the framework, because he capture-playback tool suffered from instability. As a result of that the framework needed constant maintenance and supervision.

After completing my internship I suggested that I could do my thesis on the subject of designing an automated module testing framework with the express intent of creating a stable module testing environment. Experience gained from the previous implementation taught that the new framework should be designed with these key principles in mind:

- No reliance to any specific capture-playback tool
- Framework must be scalable
- Easy test management
- Low maintenance

The purpose of this thesis was to design specification for module test automation framework that could later be implemented.

Project deliverables:

- Automated module testing framework specification
- Project plan
- Project folder

- Thesis

1.2 Terms and conventions

Table 1-1 presents the key terms used throughout this document.

Table 1. Terms

Term	Explanation
ASTF	Automated Software Testing Framework
Code-coverage testing	Code coverage testing is used to find out how much of the software's code is tested.
DB	Database
GUI	Graphical user interface.
Module	A single separate logical part of the software.
Software back-end	Refers to the software's business logic and data-stores.
UML	Unified Modeling Language is used to visually express object-oriented systems.

1.3 Company presentation

Tellabs Inc. was established in 1975 in USA. In year 1993 Tellabs, Inc. bought Martis Oy and founded Tellabs Oy (Tellabs 2002). Tellabs Oy currently operates in three locations, two in Espoo and one in Oulu. The company employs around 500 persons.

Tellabs Oy develops communication equipment and related software, focusing on mobile backhaul solutions, optical networking and intelligent network management software. The company invests heavily on research and development. In 2010 the monetary investment into R&D was little less than half of overall operating expenses (Tellabs 2010).

2 Software Testing

2.1 Overview

The role of testing in software development is to find defects (Haikala & Märijärvi 2006, 40). Testing has crucial role in software development in determining the software quality. It can consume over 50% of the overall development schedule (Dustin, Garett & Gauf 2009, 26-27).

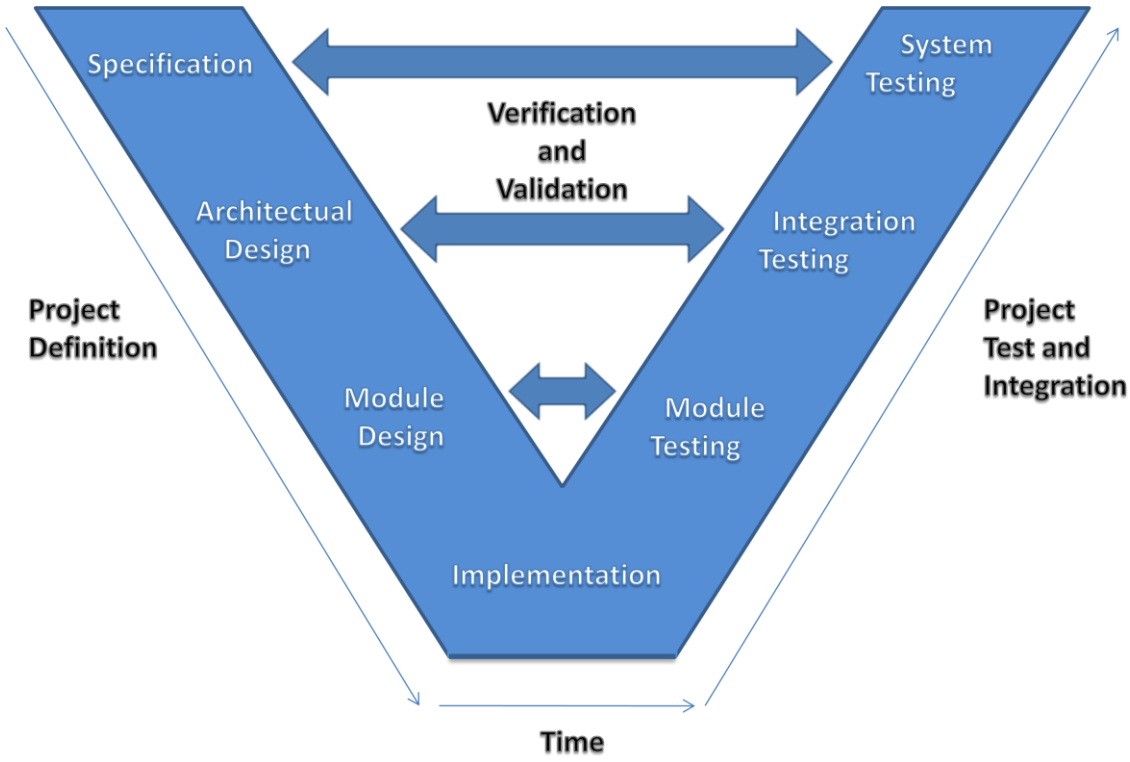

Figure 1. V-model

The V-model present in Figure 1 represents different levels in software testing (Haikala & Märijärvi 2006, 288-291). It shows what testing method is used to verify what project definition. E.g. module testing is used to verify the results done in module design (see chapter 2.3 Module testing). This thesis concentrates on the aspects of module test automation.

2.2 Business case

Because software testing is integral part of software development, it directly affects development expenses. Cost of fixing defects rises as the software progress in the development. Defects that are discovered by customers from deployed software are the most costly to fix. (Dustin, Rashka & Paul 1999, 8)

This can also be seen on Figure 2 which shows that the cost of fixing a defect in post release is more than hundred times more costly than fixing them in the design and code phase (i.e. implementation). Though the pressure to meet delivery deadline can limit the amount of testing or cause removal of key-features (Dustin, Rashka & Paul 1999, 3-4).

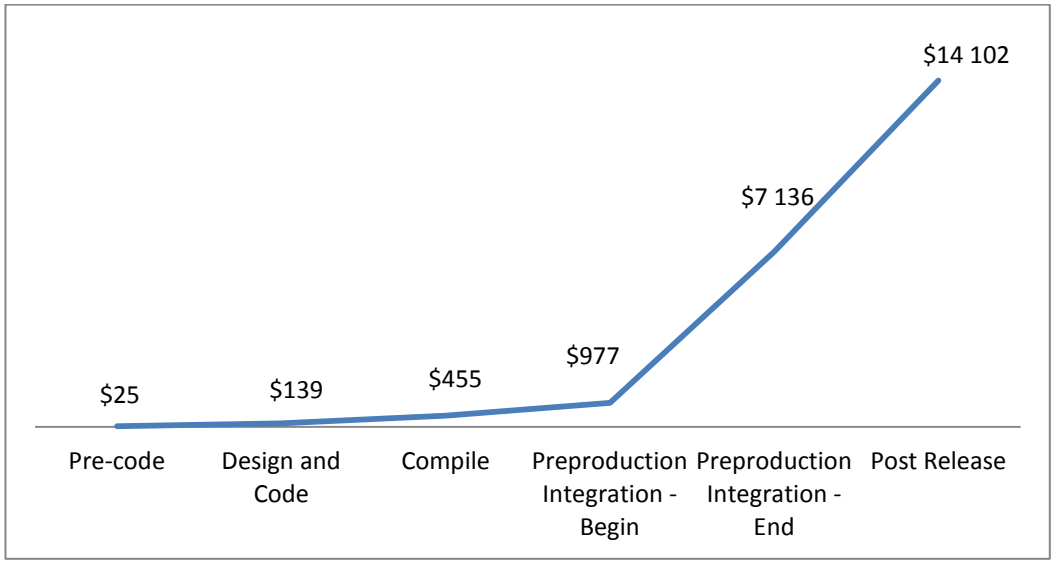

Figure 2. Defect correction costs in software development (Dustin, Garrett & Gauf 2006, 173)

Monetary loss is one aspect of the business case, but when defects emerge in critical system the results can be catastrophic. A report commissioned by National Institute of Standards & Technology stated that failures due to poor quality in aerospace products resulted in loss of life during 1993 to 1999 (2002, 35).

Fewster and Graham (1999, 5) state that once test automation is implemented the cost and effort needed to run tests is a fraction compared to manual testing.

2.3 **Module testing**

Module testing is considered a low-level testing. This means that it focuses on testing modules in the system rather than the software as a whole. It requires some knowledge of the internal structure of the software. Because it is one of the first testing phases to be done after the implementation, it is usually done by the developers that implement the modules. (Koomen & Martin, 2003. 11-13)

Software developers execute a series of steps to the target application to verify that the implementation of the module works as specified in the module design phase. Table 2 presents a simple module test case for testing Calculator applications add function.

Table 2 Calculators module test example

Step	Action	Expected result	Test result (pass/fail)
1	Open Calculator	Calculator window opens and displays "0,"	Pass
2	Write "123" using num-pad.	Calculator displays "123,"	Pass
3	Press "+" button	No visible change in the application.	Pass
4	Write "321" using num-pad.	Calculator displays "321,"	Pass
5	Press "=" button.	Calculator displays "444"	Pass

In the test specified in table 2 the verification is done by looking at the calculator's text field and comparing the number with expected result. This can be called "black box" testing, where test can be verified from the graphical user interface (i.e. GUI) without the knowledge of the software's internal structure. It shows that testing critical a part of the software can be simple, but problems arise when calculators add function needs to be tested with tens or even hundreds of permutations to ensure adequate software

quality. These permutations could include testing with decimal, negative, percentage and different length values (Koomen & Martin 2003, 17-23). Running these kinds of tests using software developers is expensive, time consuming and extremely repetitive (Fewster & Graham 1999, 9-10).

Table 3 Product module test example

Step	Action	Expected result	Test result (pass/fail)
1	Open Product management software	Product management window opens.	Pass
2	Click "Add new product"	Add new product dialog appears.	Pass
3	Write "Banana" to the 'Product name' -text field.	'Product name' -text field displays "Banana"	Pass
4	Write "A fruit" to the 'Product info' -text field	'Product info' -text field displays "A fruit"	Pass
5	Press 'Add new product to database' button	'Product added' displayed.	Pass
6	Verify that the database contains new record with name 'Banana' and Info 'A fruit'.	Product is found in the database.	Pass

Table 3 shows another module test that adds new level to the verification of the test results. The software under testing is an imaginary product management application that uses database to store information about the products. The result of the test has to be verified from the database otherwise the test cannot guarantee test quality of the particular module. This is called "gray box" testing, where the test executor has to have some knowledge about internal component (Dustin, Garrett & Gauf 2009, 12-13).

2.4 Software test automation

Software test automation term encompasses all types of testing from module testing to system integration testing. Test-driven development relies heavily on automated soft-

ware testing (Müller & Padberg 2003, 1). This development approach concentrates on developing tests for the software features before any actual feature implementation.

Effective automated software testing supports software that runs on multiple computers and different operating systems, is developed using different programming languages and either has a GUI or not. (Dustin, Garett & Gauf 2009, 4-5)

2.5 Challenges in software test automation

Most of the challenges linked to software test automation are based on the presumption that the whole testing process could be automated. This can lead to overly optimistic expectations when starting to implement software automation. Software automation also is not perfect and can still miss defects in the software under testing. Software test automation also requires maintenance like any other software. Poorly constructed automated test can cause unnecessary maintenance work and even cause the abandoning of test automation. (Fewster & Graham 1999, 10-11)

Challenges also appear on the definition of the scope that test automation can be applied. Dustin, Garrett and Gauf (2006, 26-28) state that around 40% to 60% of the tests done in most projects could be automated.

2.6 Tools for software test automation

Software test automation is based on using tools that automate some part of the software testing process (Fewster & Graham 1999, 5-6). The next chapters introduce one of these tools and a framework that utilizes it.

2.6.1 Capture-playback

Capture-playback tools are designed to automate GUI testing with users first "recording" the test. The tool tracks what buttons user presses and what text user inputs to the GUI. Most capture-playback tools use a scripting language to express the content of a test. (Fawker & Graham 1999, 43-45)

Using the example provided in Table 2, user would execute the steps with capture-playback software recording, to create test that could be run with the tool. Table 3 shows pseudo-code example that could be generated when running the simple calculator test through capture-playback tool.

Table 4 Pseudo-code script for Calculator example

Row	Command
1	LeftMouseClick 'Calculator'
2	FocusOn 'InputField'
3	Type '123'
4	Press '+'
5	Type '321'
6	Press '='
7	Does 'InputField' equal '444'

Problem with this approach is that there needs to be as many scripts as there are permutations of the test. Even if test could be run as a single batch, maintaining large test pool becomes increasingly complex and labor-intensive when software gains new features that have to be taken into consideration in tests.

Table 5 Commercial capture-replay tools script for Calculator test case example

```
Window("Calculator").Activate
Window("Calculator").WinEdit("Edit").Type "123"
Window("Calculator").WinButton("+").Click
Window("Calculator").WinEdit("Edit").Type "321"
Window("Calculator").WinButton("=").Click
If Window("Calculator").WinEdit("Edit") Is "444" Then
        test = true
Else
        test = false
End If
```

Table 5 presents the Calculator in script for that is created using a commercial capture-playback tool. In the script the verification is done be reading the edit field and comparing the argument with hard-coded value "444".

Maintenance also becomes issue when adding new features to software or changing how the program responds. This is because strict procedural test execution cannot handle situations where programs flow has changed, e.g. in Table 5 '+' changes into 'add' (Fewster & Graham 1999, 66-70).

Because capture-playback is mainly used to access the front-end of the application the example described in table 3 in hard to verify, unless the software provides some sort of front end to the database or capture-replay tool provides an interface that could access the database. Experiences gained from the previous implementation showed that using capture-replay to verify information is possible, but due to the instability of the software but not feasible.

2.7 **Automated Software Testing Framework**

Automated software testing framework is intended to create abstraction layer between tests and testing software (e.g. capture-playback tool). Depending on its scope, the framework can support one or more of the following testing methods (Dustin, Garrett & Gauf 2009, 150-151):

- Code coverage
- Black-box testing
- Grey-box testing

Dustin, Garett & Gauf (2009, 4-11) state the following key principle for automated software testing framework:

- Testing with should be possible on multiple computers concurrently
- Framework supports different operating systems or platforms
- Supports new and existing applications
- Enables test reuse

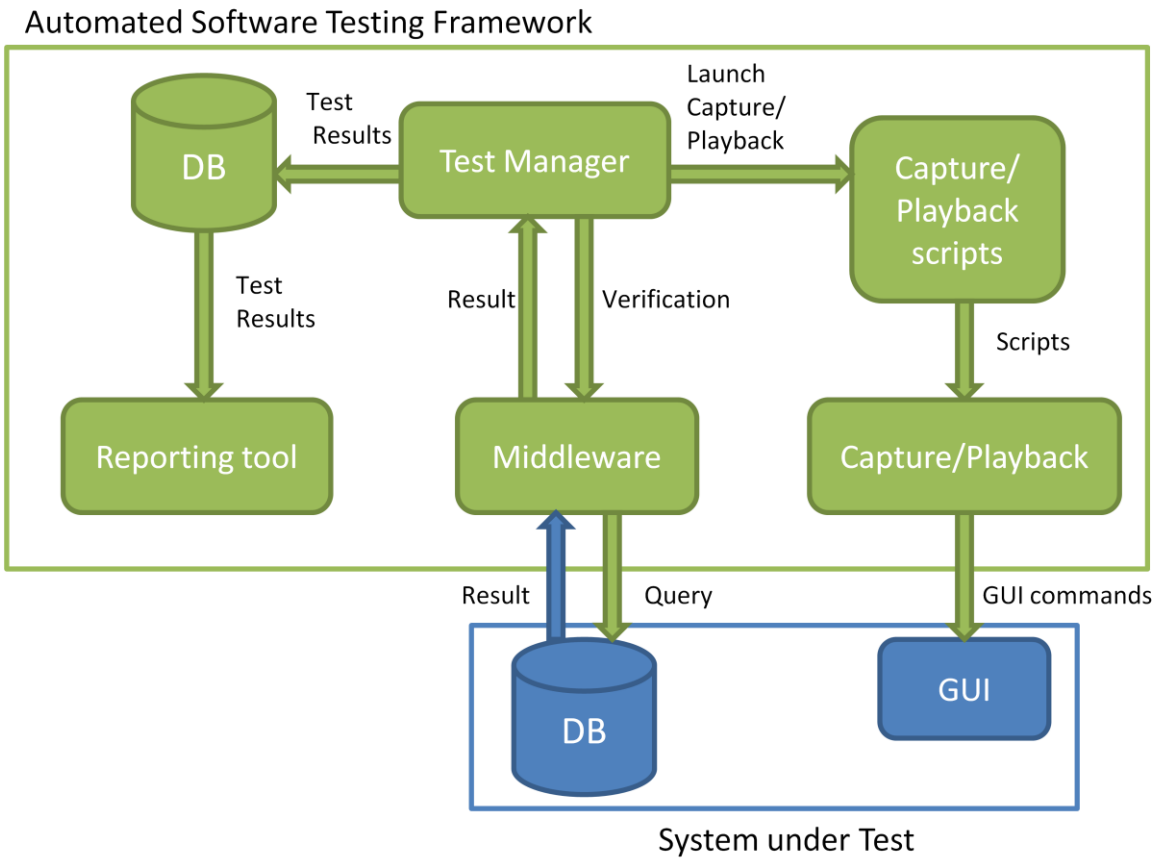

Figure 3. Automated Software Testing Framework example

The above example show what automated testing framework could contain and how the internal modules would interact. In the example test manager is responsible for start test runs using capture-playback software. Capture-playback software executes the GUI commands to the target software (i.e. system under test), the tool is not tasked to do any verification in this example.

That is the responsibility of the "middleware" module. It can execute queries to the target software's database and reports the finding to the test manager (see Table 3. Product module test example). Because this is a separate component from the test manager, it could be extended to verify the test results from other resource, such as network nodes.

Test manager handles the test run and stores the results in to the framework's database. The database is then used be the reporting tool to generate reports and statistics of the tests.

Even this fairly simple example of automated software testing framework provides tools which combine the effectiveness of capture-playback tools and still allow the test to be verified from software's back-end.

Such automated software testing framework should be separated from other environments, e.g. software development environment, to provide a stable testing environment (Dustin, Garrett & Gauf 2009, 150-151).

3 Automated module testing framework design execution

The next chapters describe the execution of designing automated module testing framework.

3.1 Project plan

Project plan was created to establish goals, scope, estimates, analyze risks and introduce the project organization.

Goal of the project was to design an automated module testing framework that would suit the need of the employer. Scope was limited to the production of functional and system architecture specification documents (see Appendix 1. Project plan).

Project organization included thesis mentor assigned by HAAGA-HELIA, project manager from Tellabs Oy, who also acted as the thesis project manager, and I acting as head designer. The head designer was responsible for executing the project assignment. Mentors were responsible for providing support to the head designer in project related matters. Project manager's responsibility extended from mentoring to guiding and supervising the project progression.

Project risks were elicited and analyzed at the beginning of the project and were specified after functional specifications.

Project started on schedule 29.4.2011 and was ended in the final project meeting 6.9.2011. Project was estimated to last for 95 days, but finished four work days ahead of schedule. One day was calculated to contain 5 hours of effective project work.

3.2 Functional specification

Functional specification started on the gathering of requirements for the framework. Key principles for the new framework were gathered from earlier implementation by rationalizing structure and architecture. This lead to the establishment of framework's key terms and conventions. (see Appendix 1. Project Plan).

Meetings were held to discuss with stakeholders about the nature of the automated module testing framework. Discussion ranged from the structure of a module test to be supported by the new system to GUI in which users would create the tests. Some requirements were specified after open dialog with the stakeholder to get clearer image of need for the specific requirement.

This phase of the design process suffered from feature creep, the amount of requirements gathered from stakeholders and from rationalizing the earlier implementation was considerable. The impact of the risk was augmented the fact that most feature requirement took considerably longer to elicit (see chapter 5.1 Project execution).

The framework structure specified in the project plan was used the divide the requirements per module basis. One aberration was done from the normal functional specification process, which was that the framework's database was planned early in the functional specification phase. This helped to establish a clear picture of the frameworks structure early in the project.

3.2.1 Specification review

The functional specification was quality checked at a specification review and corrected according to comments. After the specification was reviewed the project progressed to system architecture specification.

3.3 System architecture specification

The first goal of the system architecture specification was to determine responsibilities of the frameworks modules. The scope change affected positively on this phase of the design process.

Architecture specification was created using bottom-up approach in the design of systems internal modules. The reason for this approach was that the capture-replay tool, being the bottom-most module in the framework, was a familiar tool. Its use in the earlier implementation made it easier the write technical specification for it.

Requirements for modules were already specified in the functional description, so the process of designing the modules was relatively simple. The internal structure of the framework was compiled of components that had clear and distinctive responsibilities. Component coupling was avoided when possible to ease the development and implementation of new features.

Database design was modified when new requirements emerged that needed a specific data to be stored. This was a continuous process that lasted for the entire project.

4 Work methods

The next chapter introduces work methods used during the project.

4.1 Documentation

Tellabs documentation guidelines where used in the specification and project documentation. Documentation differed from HAAGA-HELIA's guidelines because Tellabs documentation guidelines provided the same level of detail and it supported the continued development of the framework within the company.

4.2 Meetings

During the project meeting were held in which stakeholder gather with the head designer to discuss on the state of the project and give feedback. Participants were encouraged to an open discussion on the subject of the meeting.

4.3 Face-to-face discussions

Face-to-face discussion where to use to gather requirements, observe current manual testing methods. Input from software developers that execute manual tests to the software was crucial for the functional requirements. Face-to-face discussions lasted usually for 10 to 20 minutes, but sometimes stretched to one hour.

4.4 UML

UML was used to visualize the frameworks technical structure in system architecture specification. Sequence diagrams were used to visualize the behavior of the system components.

4.4.1 Use cases

Uses cases where constructed in the functional specification phase of the project. These where at maximum two pages long with described goals, actors, pre-requisites and post-conditions.

5 Results

The next chapter describes the project execution and deliverables.

5.1 Project execution

The project was completed ahead of schedule. It suffered from feature-creep that caused change in projects scope. The initial scope included implementation, but because functional specification phase lasted three times longer than originally planned.

After the change in project scope system architecture phase was started (see Appendix 2. Final report).

5.2 Framework architecture

Existing module testing framework served as a starting point for the new framework design.

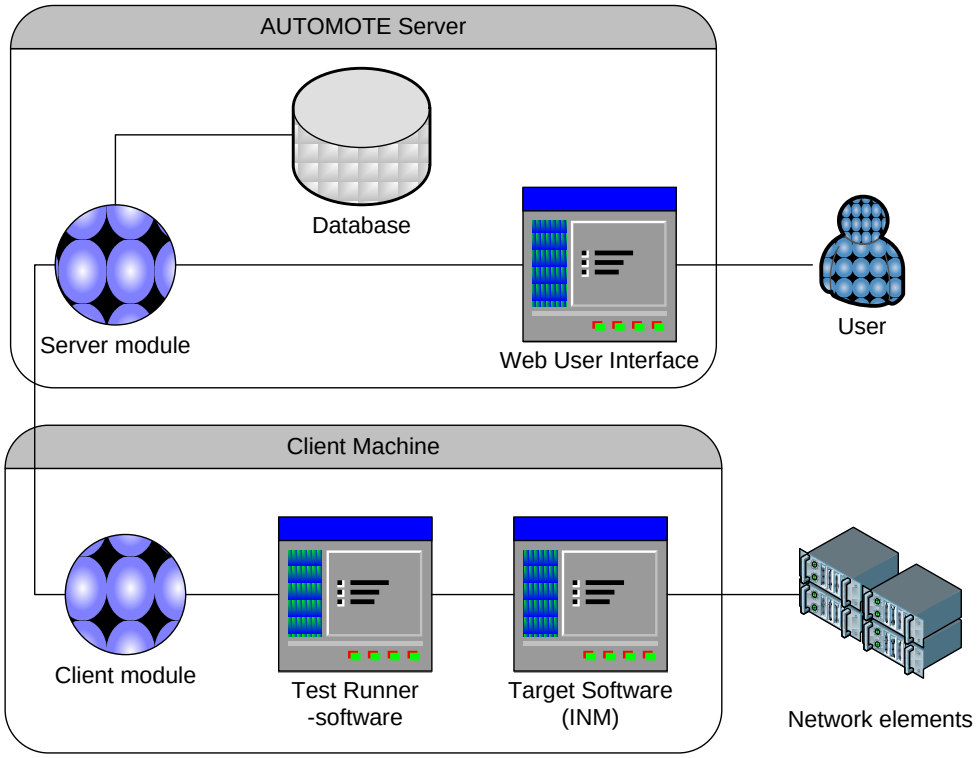

Figure 4. High-level architecture plan for the new automated module testing framework

The framework architecture were divided into three main modules, database and test runner. These modules were server, client and web user interface.

5.2.1 Server

Based on the experience gained from the earlier implementation, a central test manager was seen as a useful part of the system. Its' main responsibilities will be to communicate tests and test results with clients. Server also is the module only that has direct

access to the database. This helps aggregate the database queries and ensure that the database is stable.

The most beneficial role that the server provides is the separation of test runs from a centralized runner to multiple "testing clients" that would run the given test. In practice this allows multiple computers to test the target software simultaneously, which reduces the time spent testing e.g. one software build.

5.2.2 Client

Client was design to be a middleware between the server and test runner (see Figure 3). The need for a middleware module between the server module and the capture-playback tool stemmed from the need to verify tests from network elements and other data sources which were unreachable by the capture-playback tool. Its responsibilities included receiving tests from the server and running them with the capture-playback tool. Client can be compared to the test manager shown in table 3.

Client provides the benefit of being the abstraction layer between the capture-replay tool and the server. If the company decides to change to provider for the capture-replay tool or decide to develop one in-house, the client reduces amount of work that needs to be done in such events to the framework.

5.2.3 Web user interface

The framework needed to provide users with a flexible GUI with which to manage the framework, generate reports and develop tests. It was decided that a web based module that only needed a basic web browser from user would be the most ideal solution.

Main benefit of this module was that users wouldn't need to install any new software in order to create automated module tests and it would allow rapid deployment of new features depending on the implementation web framework.

5.2.4 Capture-replay tool

Capture-replay tool (also referred as test runner in Figure 4) is software provided by an external company and uses proprietary scripting language. As noted from the previous implementation it also suffered from instability. This limited the usability of the tool to mere running GUI commands to the target software.

5.3 Project deliverables

Project deliverables included a project folder that contained project plan, final report, functional and system architecture specification and project communication transcripts.

6 Discussion

The next chapter discusses the thesis conclusions and suggestions for further development.

6.1 Conclusions

The project goals to design an automated module testing framework and produce functional and system architecture specification were met. The functional and system architecture specification was deemed confidential by Tellabs Oy. Project scope had to be changed in the middle of the project. It proved to be beneficial by allowing more time to be spent on the system architecture specification. The project organization reacted to the scope change and the project plan was updated to reflect the new situation.

Open dialog communication proved to be effective way to convey what the stakeholder were expecting of the system. Even though disagreements rose between stakeholders on some details of the system, the head designer must weigh each argument and make a decision based on the knowledge at hand. Defining of terms and conventions for the framework was found to be important to enable clear and concise project communication.

Clear and concise documentation was found to be essential when designing the software product. The functional specification document allowed stakeholders to engage the design process commenting on the system based on the use cases and functional description. Most of these comments were addressed at latest in the functional specification review.

Designing the database early in the project had positive effect on the project as a whole. Stakeholder could immediately see what information was stored and how it was linked. This allowed them to form a concept of the framework early in the design process which in return resulted in much needed feedback. Visualization in general proved to be powerful conveyor of ideas and much effective than mere textual one.

6.2 **Suggestions for further development**

It is suggested that the Tellabs Oy continues with the development of the automated module testing framework by starting the framework implementation. Even though automated software testing does not provide a short-term improvement to module testing, the long-term effects include savings in operational costs and rise in software quality (Dustin, Garrett, Gauf 2009, 26-42).

References

Booch, G., Jacobson, I., Rumbaugh, J. 1999. The Unified Modeling Language User Guide. Addison-Wesley. United States of America.

Dustin, E., Garrett, T., Gauf, B. 2009. Implementing Automated Software Testing. Addison-Wesley. United States of America.

Dustin, E., Rashka, J., Paul, J. 1999. Automated Software Testing. Addison-Wesley. United States of America.

Fewster, M., Graham, D. 1999. Software Test Automation. Addison-Wesley. England.

Haikala, I., Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Talentum. Finland.

Haikala, I., Märijärvi, J. 2006. Ohjelmistotuotanto. Talentum. Finland.

Hofmeister, C., Nord, R., Soni, D. 2000. Applied Software Architecture. Addison-Wesley. United States of America.

Jaaksi, A., Aalto, J-M., Aalto, A., Vättö, K. 1999. Tried & True Object Development. Cambridge University Press. England.

Kit, E. 1995. Software Testing in the Real World. Addison-Wesley. United States of America. q

Koomen, T., Pol, M. 2003. Test Process Improvement. Addison-Wesley. United States of America.

Müller, M., Padberg, F. 2003. About the Return on Investment of Test-Driven Developmen. Source: http://www.ipd.uka.de/mitarbeiter/muellerm/publications/edser03.pdf Retrieved: 5.9.2011.

National Institute of Standards & Technology 2003. The Economic Impacts of Inadequate Infrastructure for Software Testing. Source: http://www.nist.gov/director/planning/upload/report02-3.pdf Retrieved: 5.9.2011.

Schmidt, D., Stal, M., Buschmann, F., 2001. Pattern-Oriented Software Architecture Volume 2. Wiley. England.

Tellabs annual report 2010, Source: http://www.tellabs.com/investors/annual/2010/tellabs10annual.pdf Retrieved 18.8.2011.

Tellabs. 2002. Tellabs Suomessa. Brochure.

Appendices

Appendix 1. Project Plan

Appendix 2. Final Report

AUTOMOTE FRAMEWORK

Project plan

v2.4

TABLE OF CONTENTS

1 Document Information

This document describes the scope, responsibilities, resources, milestones and deliverables for the AUTOMOTE Framework project and has references to other related documents.

1.1 Version history

Author	Version	Date	Comments
Jukka Huiskonen	1.0	27.4.2011	Initial version.
Jukka Huiskonen	1.1	27.4.2011	Fixed with comments from Henri Laamanen
Jukka Huiskonen	1.2	4.5.2011	Fixed with comments from project meeting
Jukka Huiskonen	2.0	7.5.2011	New version updated with comments from project organization.
Jukka Huiskonen	2.1	9.6.2011	Updated project schedule.
Jukka Huiskonen	2.2	2.8.2011	Revised project goal and schedule
Jukka Huiskonen	2.3	15.8.2011	Updated project risks and removed redundant milestones.
Jukka Huiskonen	2.4	2.9.2011	Updated to with the latest project information.

Note: The date format used throughout this document is: <dd>.<mm>.<yyyy>.

1.2 References

Ref.	Document / Site Name	Path
N/A		

1.3 Other information

1.3.1 List of figures

Terms and conventions

Term	Explanation
AUTOMOTE	Automated Module Testing
CI	Continuous Integration
DB	Database
DSL	Domain Specific Language
INM	Intelligent Network Manager
NMS	Network Management System
UI	User interface
WHRT	Work Hour Reporting Tool

2 Goals and Objectives

This document is project plan of automated module testing framework design project.

2.1 Project Goal

Goal of the project is to design automate module testing system by rationalizing and refactoring current test system architecture.

2.2 Impact

Impact of automated module test is savings in operation costs and increased quality of software. Automated module testing framework runs every implemented test after build is changed, so that every tested feature is retested in every build. Automation also lowers the man-hours needed in manual module testing.

2.3 Current status

Current test system has several disadvantages, it consists of three separate codebase project; web interface, test software code and client side software. Web interface is implemented using mix of PHP and JavaScript. MySQL is used as database to store test results. No frameworks are used. Client side software starts test software and takes care of build installation and setting up test environment. It is written in Java and does not use any frameworks. Test software has function libraries written in VBS, functions have no formal pattern, and also logic that queries new test from the server.

This approach results in high maintenance cost and unreliability of the test results (false positives and test software crashing).

2.4 New test system architecture

The new test system architecture consists of three distinct parts; Server, Client and Web interface.

Server will store, and instruct robots to run, tests and reports in a database. The server will also house web interface which is used to maintain the test framework, build new tests, gather reports and control robots.

Client side works as interface to the test software, it gets test from the server and instructs the test software to run them and reports the result to the server. Robots handle installation of new builds, resetting the environment and reporting of faulty builds.

Server keeps track of clients (status, build, processes etc.) and handles rising error statuses with predetermined rules. E.g. if build installation fails, try installing it n times again, if it still fails to install, mark build defective and request new build or retrieve it automatically.

Web interfaces is the main interface that user access the test framework. All tests are done through the web interface and stored to the DB through the server for “version” control. Tests are first marked as “in development” to stop them from interfering with testing environment in case they contain errors. When “in development” test passes without errors in the development environment it can be upgraded to “published” and run in the real test environment. Reports are gathered from every step of the test and reported to the server.

2.5 Project scope

The scope of this project is to design four modules: client-, server module, web interface and test runner. Web interface works as a reporting and maintenance interface to the framework. Server module notifies the client module of new test and receives test results from it. Client module runs the test it receives to test software and monitors the client machine. Test runner is a program that runs tests scripts. The specification includes only script implementation that enable the singular test steps to be tested in target software.

- Automated module test framework for INM
    - WHRT: SM_AUTOMOTE
        1. Feature: SM_AUTOMOTE
        2. Project: Service Management
        3. Release: None
    - Modules and responsibilities
        1. Client side: Jukka Huiskonen
        2. Server side: Jukka Huiskonen
        3. Web UI: Jukka Huiskonen

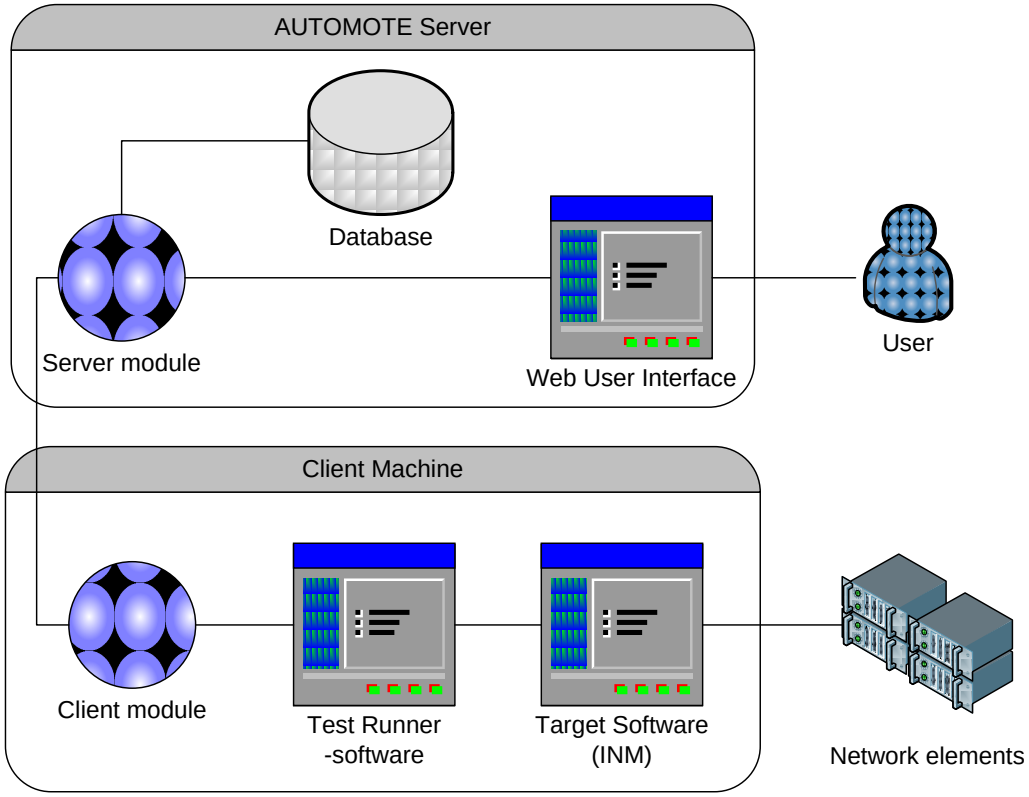

Figure 1 AUTOMOTE Framework Client/Server scope

2.6 Work estimates

Work estimates of the project are presented in the table below.

Task	Effective Man days (= 5 hours)		
	Work Done	Work Left	TOTAL
Specification	83	0	83
Other tasks	5	0	5
Project Management	2	0	3
Summary/Days	**91**	**0**	**91**

2.7 Project organization and responsibilities

The project is a Sub-project of Service Management project. The project decides independently about use of resources and sharing of tasks inside the project. Jukka Huiskonen reports to Service Management project manager and Project Mentors.

Project team:

Person	Role	Known absences
Henri Laamanen	Service Management Project Manager, Project Mentor	
Jyri Partanen	Project Mentor	
Jukka Pekka Huiskonen	Head Designer	11.04.-12.04. NMS Training

2.8 Main phases and milestones

Milestone	Original Estimate	Current estimate	Actual
Specifications and plans ready			
▪ Project plan ready	27.04.2011	06.09.2011	06.09.2011
▪ Specification ready	31.05.2011	02.09.2011	02.09.2011

3 Managerial Process

3.1 Meetings

Project meetings are held at the start of the project, midway and at the end of the project. These meetings are scheduled by Jukka Huiskonen to fit the Project Managers and Project Mentors schedules. Project status is also covered twice in a week in Service Management project meetings, Tuesdays and Thursdays.

3.2 Reporting status

Status will be monitored using the WHRT, project meetings and face-to-face discussions as needed. Feature statuses will be updated to intranet site.

3.3 Escalation process

Project reports to Service Management project. If an issue cannot be solved within the project, it will be escalated to manager of Service Management project. If an issue affects on timetable or is seen otherwise as critical, it will be escalated to Service Management project immediately. In other cases issues are dealt inside the project using internal priorization.

3.4 Communication

All forms of communication from informal discussions to formal reviews are used. The project members are encouraged to an open dialog.

3.5 Processes and deliverables

This project follows waterfall style software development cycle. The size of the project organization allows some departure from the traditional waterfall model; to cope with the rise of new specification and requirements. Deliverables are described in this document (high-level architecture) and in specification (low-level technical architecture).

3.6 Documentation

All electronic copies of the project document, and information generated electronically will be stored to project's intranet site.

Project status can be seen from the projects feature status sheet that will be made after specification document is ready. The document will be stored to the same location as project plan.

Specifications will be added to the same location as project plan when

3.7 Tools

Project status reports	MS Office Power Point 2007
Plans, specifications, reports	MS Office Word 2007, Intranet
Work hour reporting	WHRT
Requirement management	Word

3.8 Training

Jukka Huiskonen: NMS Training 11.05.-12.05.2011 (SK7)

4 Risks

Risk (what may go wrong)	Drivers (why is this a risk, what makes it more probable)	Mitigation strategy (how to prevent this)	Severity (H,M,L)	Probability (H, M, L)	Owner
Maintenance of the current AUTOMOTE framework	Current framework may need maintenance	Priorization, escalation to service management project	M	H	Service Management, project manager
Unexpected extra work due to something unrelated	Possibilities of process improvement in build testing.	Priorization	M	M	Service Management, project manager
Feature creep	Desire to produce all-encompassing testing framework.	Document essential features first and design system to be expandable.	M	H	Service Management, project manager

5 Quality

5.1 Verification of deliverables

The following deliverables will be reviewed using formal review process: Functional and System Architecture Specifications and Project Plan.

6 Related Projects

Role	Person/board
Service Management Project	Henri Laamanen

7 Needed Hardware

No need for new hardware.

AUTOMOTE FRAMEWORK

FINAL REPORT

Jukka Huiskonen

2.9.2011

TABLE OF CONTENTS

DOCUMENT INFORMATION

Scope

This document describes background, progression, results, experiences and suggestions for further development for AUTOMOTE framework.

Version history

Author	Version	Date	Comments
Jukka Huiskonen	1.0	2.9.2011	Initial version.

References

Ref.	Document / Site Name	Path
[FS&SAS]	AUTOMOTE Functional and system architecture description	

Terms and conventions

Term	Explanation
AUTOMOTE	Automated Module Testing

Introduction

AUTOMOTE Framework project was executed between 29.4.2011 - 8.9.2011. This was executed as a thesis project which goal was to create a design specification for automated module testing framework. The scope had to be adjusted in the middle of the project because feature creep risk actualized and impacted projects on the timetable (see chapter 3.6.5 Project evaluation).

1 Project goal

Goal of the project was to design automate module testing framework and produce a functional and system architecture description that could be used to implement the system.

2 Project description

The next chapters describe the projects scope and work estimates.

2.1 Project scope

Projects scope was to design a functional and system architecture specification for the automated module testing framework.

2.2 Work estimate

Work estimates of the project are presented in the table below. One workday contains 5 hours of effective project work.

Table 1. Work estimates

Task	Work estimate	Implementation weeks
Specification	87	17-35
Functional Specification	40	17-26
System Architecture Spec.	47	27-35
Other tasks	5	17-36
Project Management	3	17, 24, 36
Total	**95**	**17-36**

2.3 Project documentation and quality control

The next chapter describes the quality control used in the project as well as documentation and used tools methods.

2.3.1 Project follow-up documentation

Project organization gathered three times (see Table 2. Project meetings). Meeting notices where sent at least three days before each meeting. Meeting minutes were compiled during the meeting

and sent to everyone in the project organization within three days of the meeting. Meeting minutes where reviewed and accepted in the next meeting.

2.3.2 Used tools and method

Requirements gathering were done using meetings and face-to-face discussions with the stakeholders.

Documentation was created using Microsoft Office 2007 products. Tellabs documentation practice was used in this project. Versioning was done by adding version number to the document filename. Project management documents had only date they were either created or modified as a prefix. Some managerial documents as project plan included version details inside the first chapter.

Tellabs intranet was used to store the documents because of its internal backup.

2.3.3 Quality control

Specification review was used as a quality gate for the project. Specification was corrected according to the suggestions presented in the specification review.

Each project organization meeting reviewed the proceedings of last meeting by accepting the meeting minutes.

2.4 Managerial process

In the next chapter the managerial process of the project is described.

2.4.1 Roles and responsibilities

Project organization consisted of thee persons; one project manager who also acted as a project mentor, another project mentor assigned by HAAGA-HELIA and a head designer that executed the project.

Project manager was responsible supervising and guiding the project. Mentors advised the head designer on the project. Head designers responsibility was to execute the project and notify the rest of the project organization of the project's progress.

2.4.2 Project meetings

Project development was evaluated at the following meetings:

Table 2. Project meetings

Meeting	Result	Date
Project kick-off	Approved project plan.	29.04.2011
Project follow-up	Meeting minutes, evaluated follow-up report and accepted functional specification.	17.06.2011
Final meeting	Approved final report	6.9.2011

2.5 Project work

In this chapter the work process of the project is described

2.5.1 Meetings and discussion

Meetings with the stakeholders were hold during the functional specification phase. Meeting focused on the general aspects of the functions that the system must provide. Open face-to-face discussions were used to get more specific information on the required functionality of the system.

2.5.2 Functional specification

Functional specification was built on the requirements gather from stakeholders. The earlier implementation was used as a comparison point to systems structure, which was already in the project plan (framework architecture).

2.5.3 Spefication review

The functional specification review was done to ensure that the system when implemented would provide the necessary functions that stakeholders need.

2.5.4 System architecture

System architecture was designed by eliciting the responsibilities of each module in the system, and then moving to design the module's internal structure. The approach taken with the system was bottom-up. This was because the requirements for the lowest parts of the system where the most clear and simple.

2.6 Project deliverables, evaluation and suggestions

2.6.1 Deliverable documentation

The project produced the following documentation

- AUTOMOTE Framework Functional and System Architecture Specification
- Project folder

Project folder contains all project related documentation.

2.6.2 Achieved work

Achieved work of the project is presented in the table below.

Table 3. Achieved work

Task	Effective Man days (= 5 hours)		
	Estimated	Work done	Difference (%)
Specification	87	83	-5%
Other tasks	5	5	0%
Project Management	3	3	0%
Summary/Days	**95**	**91**	**-5%**

Project was executed four work days ahead of schedule; this was due to the fact that system architecture specification was completed faster than planned.

2.6.3 Risks realization

Table 4. Project risks

Risk (what may go wrong)	Drivers (why is this a risk, what makes it more probable)	Mitigation strategy (how to prevent this)	Severity (H,M,L)	Probability (H, M, L)	Owner
Maintenance of the current AUTOMOTE framework	Current framework may need maintenance	Priorization, escalation to service management project	M	H	Service Management, project manager
Unexpected extra work due to something unrelated	Possibilities of process improvement in build testing.	Priorization	M	M	Service Management, project manager
Feature creep	Desire to produce all-encompassing testing framework.	Document essential features first and design system to be expandable.	M	H	Service Management, project manager

Feature creep was the only risk that actualized (see Table 4. Project risks). This happed in the functional specification phase of the project and lead severely impacted the project (see chapter 3.6.5 Project evaluation). The reason for the risk to actualize was the inexperience of the head designer to limit the scope of the functional specification and to decide some key principles of the system. The risk was dealt with dropping the implementation from the project and prioritizing the core features of the system as requirements and other features with "future" or "nice-to-have" priority.

2.6.4 Project budget

Project had no defined budget as this was done as a thesis project.

2.6.5 Project evaluation

Project was completed a little less than a week ahead of schedule, but still had a medium severity risk actualized. One of the reasons for the project being completed ahead of time was that all the meetings could be held as scheduled.

Actualized risk was dealt with fast response and with the mitigation strategy specified in Table 3. This caused implementation being dropped from the project's original scope. The project plan was altered to accompany this change. Though it did not have affect on the initial deadline for the project, but the project was changed from designing and implementing the system to encompass just the design part.

The scope change doubled the time to be used to design the system, which proved to be a positive change that allowed to use more time on the system architecture. The result of this was a more complete architectural specification.

Project organization acted with great care and professionalism. Mentors gave all the required information and provided support in the project managerial and design parts.

Project resulted in a functional and system architecture specification document that is going to be the basis for automated module testing framework implementation.

Learning goals for the project were met, these include:

- Complete and successful execution of the project
- Successfully gather key requirements for the project
- Understanding the design process in software development

2.6.6 Project as a learning experience

Project served as excellent learning experience into project management, gathering and managing requirements, functional and system architecture specification. Because this project first included the plan to implement the system, it really showed how a little feature creep can have larger impact on the projects timetable. It underlined the importance of good and careful project planning process. Even though not every risk can be taken in to consideration and most of the time risks actualize, working project organization and good mitigation strategy for the elicited risks are the foundations to good risk management in software projects.

The design process was affected by the earlier implementation. This proved to be crucial in designing the new system. Otherwise the design process would have started from scratch and taken considerable longer. It taught that it is good to look how the problem has been solved in other implementations.

System architecture design was mostly done by utilizing the teachings learned in HAAGA-HELIA. Some problems needed new information to be digested and applied fairly quickly. E.g. protocol that works over reliable TCP connection including the messaging syntax, proved to be interesting and rewarding learning experience.

Documentation proved to have key role on the success of the project. Specifying the scope of the project and prepare for risk should be done with great care. And in the event that risk actualizes, it must be dealt with the planned mitigation strategy as soon as possible to minimize the impact on the project.

2.6.7 Suggestions for further development

It is suggested that the development continued with the AUTOMOTE project by starting the implementation as described in the [FS&SAS].